ADDIS ABABA UNIVERSITY

ADDIS ABABA INSTITUTE OF TECHNOLOGY (AAiT)

School of Information Technology and Engineering(SiTE)

MACHINE LEARNING AND BIG DATA

PROJECT TWO (CLASSIFICATION)

NAME: Natan Amanuel Mamo

Id no: ATR/7142/13

Year: IV (Software stream)

Submitted to: Mr. Bisrat

Table of Contents

INTRODUCTION

The task of predicting whether an individual earns more than $50,000 annually has significant implications in areas such as market segmentation, public policy design, and targeted economic assistance programs. This project explores the application of machine learning classification algorithms and aims to develop a model capable of classifying individuals based on their income level using demographic and occupational features. Using a publicly available dataset, the study aims to build and evaluate models capable of predicting whether a person has an income of more than or below a certain amount. The primary objective is to employ multiple classification algorithms, including Logistic Regression, Decision Tree, Support Vector Machine (SVM), and Random Forest, to identify the most effective approach for this task.

The dataset, derived from the UCI Machine Learning Repository, contains various socio-economic attributes such as age, education, occupation, and hours worked per week. The objective is to leverage this dataset to train and evaluate classification models capable of predicting income levels.

Through this study, we aim to address the following questions:

1. Which classification algorithm provides the highest predictive accuracy and reliability for this prediction?
2. What are the most important features influencing the model's predictions?
3. How do the different algorithms compare in terms of performance metrics such as accuracy, precision, recall, F1-score, and AUC-ROC?

To achieve this, four classification algorithms were employed: Logistic Regression, Decision Tree, Random Forest, and Support Vector Machine (SVM). The models were evaluated using key performance metrics, including accuracy, precision, recall, F1-score, and the Area under the Receiver Operating Characteristic Curve (AUC-ROC). Additionally, feature importance was analyzed to understand which factors contribute most significantly to income prediction.

The report also details the data preprocessing steps applied to the dataset, the methodology used to train and evaluate the models, and concluding with findings and insights of the results. By comparing the performance of multiple algorithms, this study seeks to provide insights into the strengths and limitations of each approach for this prediction.

TOOLS AND METHODS

About the Dataset

The dataset used in this project, referred to as the "Adult" dataset, was sourced from the UCI Machine Learning Repository. It comprises census data that includes both demographic and occupational information for individuals. The dataset was designed for binary classification tasks, specifically predicting whether an individual earns more or less than $50,000 per year.

The dataset consists of more than 32,500 instances, with a mix of continuous and categorical attributes. After preprocessing and removing missing values, the final dataset used in this project contains **15 attributes**, which includes 14 features and the target variable.

Features:

1. **Age**: Continuous numeric value representing the individual's age.
2. **Workclass**: Categorical variable indicating the type of employment (e.g., Private, Self-employed).
3. **Education**: Categorical variable showing the highest level of education attained.
4. **Education-num**: Numeric representation of the education level.
5. **Marital-status**: Categorical variable indicating marital status (e.g., Married, Single).
6. **Occupation**: Categorical variable describing the type of job.
7. **Relationship**: Categorical variable describing familial relationships (e.g., Husband, Wife).
8. **Race**: Categorical variable representing race.
9. **Sex**: Categorical variable representing gender.
10. **Capital-gain**: Continuous numeric value of capital gains received.
11. **Capital-loss**: Continuous numeric value of capital losses incurred.
12. **Hours-per-week**: Continuous numeric value of hours worked per week.
13. **Native-country**: Categorical variable indicating the country of origin.

Target Variable:

1. **Income**: Binary categorical variable indicating whether the individual earns <=50K or >50K.

Data Preprocessing: Upon initial inspection, the dataset contained missing values in some categorical variables, represented by "?". These were replaced with NaN and subsequently addressed by dropping rows with missing values. Categorical variables were encoded using Label Encoding to convert them into numerical representations. Continuous features were standardized to improve the performance of algorithms sensitive to feature scaling, such as Support Vector Machines.

Class Distribution: The target variable is imbalanced, with the majority of individuals earning <=50K. This imbalance was considered during model evaluation to ensure fair performance comparison.

Jupyter Notebook

I used Jupyter Notebook for my data analysis tasks in Python. Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. It is widely used in data science and machine learning for its ability to support interactive data visualization and the integration of code, which facilitates exploratory data analysis and prototyping.

Algorithms Implemented

Classification

Classification is a supervised machine learning task that involves predicting the category or class of a given data point based on its features. In a classification problem, the algorithm learns from labeled training data to classify unseen instances into predefined categories. For this project, the task was to classify individuals into two categories based on their income: <=50K or >50K. This binary classification problem uses various machine learning models to identify patterns in the data and make predictions.

The following classification algorithms were implemented for this project:

1. Logistic Regression: Logistic Regression is a statistical method that models the relationship between independent variables and a binary dependent variable. It uses the logistic function to transform linear combinations of the input features into probabilities, which are then used to classify data points.

Why Preferred?

- Simple to implement and interpret.
- Performs well with linearly separable data.
- Provides probabilities for predictions, which is helpful for evaluating uncertainty.

2. Decision Tree: A Decision Tree is a tree-like model of decisions and their possible consequences. It splits the dataset into subsets based on feature values, creating branches that represent decision rules until a target value is achieved.

Why Preferred?

- Easy to visualize and interpret.
- Handles both numerical and categorical data.
- Captures non-linear relationships between features and the target variable.

3. <u>Random Forest</u>: Random Forest is a learning method that constructs multiple decision trees during training and outputs the mode of their predictions. It reduces overfitting by averaging predictions from multiple trees.

Why Preferred?

- Handles high-dimensional data effectively.
- Robust to outliers.
- Reduces overfitting compared to a single decision tree.
- Easy to implement next to decision tree as they are related

4. <u>Support Vector Machine (SVM)</u>: Support Vector Machine is a powerful algorithm that finds the optimal hyperplane to separate data points into different classes.

Why Preferred?

- Effective in high-dimensional spaces.
- Works well with datasets where the number of dimensions exceeds the number of samples.
- Robust to outliers due to the use of margins.

Justification for Algorithm Selection These algorithms were chosen for their diversity in approach:

- Logistic Regression provides a probabilistic baseline for comparison.
- Decision Tree offers an interpretable model for understanding feature splits.
- Random Forest introduces ensemble learning to improve robustness and accuracy.
- SVM explores margin-based classification for linear and non-linear data.

This diverse selection ensures a comprehensive evaluation of the dataset and allows for a meaningful comparison of performance metrics such as accuracy, precision, recall, F1-score, and AUC-ROC.

<u>Performance Metrics</u>

Evaluating the performance of machine learning models is a critical step in ensuring the reliability and accuracy of predictions. For this project, the performance of the classification algorithms was measured using the following metrics:

1. <u>Accuracy</u>: Accuracy is the ratio of correctly predicted instances to the total number of instances in the dataset.

Accuracy= Number of Correct Predictions / Total Number of Predictions

Accuracy is a metric that provides an overall measure of a model’s performance. However, in cases of imbalanced datasets, accuracy alone can be misleading as it may overestimate the performance of a model that predominantly predicts the majority class.

2. <u>Precision</u>: Precision measures the proportion of true positive predictions out of all positive predictions made by the model.

Precision= True Positives (TP) / (True Positives (TP) + False Positives (FP))

Precision is particularly important when the cost of false positives is high. For example, in this project, predicting a person as earning more than $50K when they actually don't could lead to inappropriate marketing or policy decisions.

3. <u>Recall</u>: Recall, also known as sensitivity or true positive rate, measures the proportion of true positives out of all actual positive instances.

Recall= True Positives (TP) / [True Positives (TP) + False Negatives (FN)]

Recall is critical when it is more important to identify all positive cases, even at the expense of increasing false positives. For example, in applications like fraud detection, missing a positive case can have significant consequences.

4. <u>F1-Score</u>: F1-score is the harmonic mean of precision and recall, providing a balanced measure that accounts for both metrics.

F1-Score= 2 × [(Precision × Recall) / (Precision + Recall)]

F1-score is particularly useful when there is an imbalance in the class distribution, as it provides a single metric that balances precision and recall.

5. <u>AUC-ROC (Area under the Receiver Operating Characteristic Curve)</u>

AUC-ROC measures the ability of a model to distinguish between classes. The ROC curve plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold values. AUC-ROC provides an aggregate measure of performance across all classification thresholds. A higher AUC value indicates that the model is better at distinguishing between positive and negative classes. It is particularly helpful in evaluating models with imbalanced datasets.

Each of these metrics evaluates a specific aspect of model performance:

- **Accuracy** provides a general measure of correctness.
- **Precision** and **Recall** address the trade-off between false positives and false negatives, making them critical for understanding errors.
- **F1-Score** balances precision and recall into a single metric, offering insight into model performance on imbalanced data.
- **AUC-ROC** evaluates the model's capability to separate the classes across thresholds, adding a probabilistic dimension to the analysis.

CODE BREAKDOWN

IMPORTING LIBRARIES

In this project, I imported several essential libraries to facilitate data manipulation and analysis. Below is a breakdown of each imported library:

- Pandas (`import pandas as pd`): Pandas is a powerful data manipulation library that provides flexible data structures, like Series and DataFrames, for complex data analysis tasks. It is widely used for reading, writing, and processing structured data efficiently, enabling operations like filtering, grouping, and aggregating.
- NumPy (`import numpy as np`): NumPy is a foundational library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, alongside a collection of mathematical functions to operate on these arrays. It excels in performing efficient numerical operations and handling linear algebra.
- Matplotlib (`import matplotlib.pyplot as plt`): Matplotlib is a plotting library that is used for creating static, interactive, and animated visualizations in Python. It provides functionalities to create various types of plots and charts, enabling data visualization to help interpret data better.
- Seaborn (`import seaborn as sns`): Seaborn is built on top of Matplotlib and simplifies the creation of attractive statistical graphics. It helps visualize complex datasets with simpler interfaces and provides enhanced visual appeal by improving the aesthetics of visualizations.
- Scikit-learn (`from sklearn.model_selection import **train_test_split**`): Scikit-learn is a comprehensive library for machine learning that offers simple and efficient tools for data mining and data analysis. The module imported is used for splitting datasets into training and testing sets, which is essential in evaluating the performance of machine learning models.
- from sklearn.preprocessing import **LabelEncoder, StandardScaler**: imports two classes from the sklearn.preprocessing module:
  - LabelEncoder: Used to convert categorical data (e.g., 'male', 'female') into numerical labels (e.g., 0, 1).
  - StandardScaler: Used to standardize features by removing the mean and scaling to unit variance.
- from sklearn.linear_model import **LogisticRegression**: Imports the LogisticRegression class from the sklearn.linear_model module.
- from sklearn.tree import **DecisionTreeClassifier**: Imports the DecisionTreeClassifier class from the sklearn.tree module.
- from sklearn.ensemble import **RandomForestClassifier**: Imports the Random Forest Classifier class from the sklearn.ensemble module.
- from sklearn.svm import **SVC**: Imports the SVC (Support Vector Classifier) class from the sklearn.svm module.

- from sklearn.metrics import **accuracy_score, confusion_matrix, classification_report, roc_auc_score, roc_curve, precision_score, recall_score, f1_score**: Imports various performance metrics from the sklearn.metrics module:
    - accuracy_score: Calculates the overall accuracy of the model.
    - confusion_matrix: Creates a matrix showing the number of true positives, true negatives, false positives, and false negatives.
    - classification_report: Generates a report with precision, recall, F1-score, and support for each class.
    - roc_auc_score: Calculates the Area Under the Receiver Operating Characteristic curve.
    - roc_curve: Computes the ROC curve, which plots the true positive rate against the false positive rate.
    - precision_score: Calculates the precision of the model.
    - recall_score: Calculates the recall of the model.
    - f1_score: Calculates the F1-score, which is the harmonic mean of precision and recall.

Each of these libraries plays a crucial role in the workflow of data analysis, ensuring the efficient manipulation, analysis, and visualization of the dataset.

READ AND ANALYZE DATA

1. Reading the CSV File (data = pd.read_csv()): This function from the Pandas library is used to read a CSV (Comma Separated Values) file. The resulting data is stored in a DataFrame. A DataFrame is a 2-dimensional labeled data structure with columns that can be of different types (similar to a table in a database or a spreadsheet).

2. Viewing the First Few Rows (data.head()): This function displays the first five rows of the DataFrame by default. It allows you to quickly glance at the data to understand its structure and content.

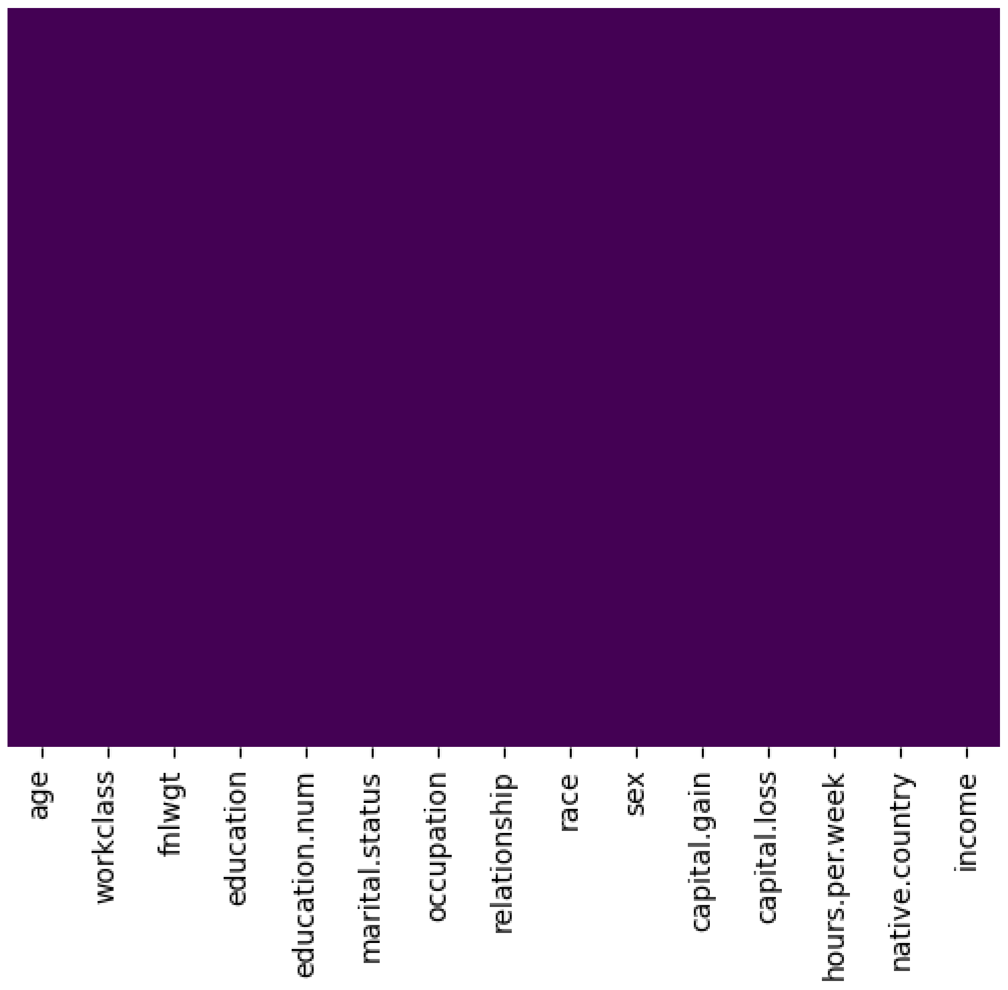
3. Viewing the Last Few Rows (data.tail()): This function displays the last five rows of the DataFrame by default.

4. Getting Basic Information About the DataFrame (`data.info()`): This function provides a concise summary of the DataFrame, including the number of entries, the data types of each column, and the number of non-null values. This is helpful for understanding the structure of the data and identifying potential data quality issues.

5. Descriptive Statistics of the DataFrame: `data.describe()` This function generates descriptive statistics of numerical columns, including count, mean, standard deviation, min, max, and quartiles. It's useful for getting a statistical summary of the data and understanding the distribution of numeric variables.

6. Observe the random sample data (data.sample(10)): This line uses the sample() method to randomly select 10 rows from the DataFrame and display them. This provides another way to explore a representative sample of the data.

7. Visualizing Missing Values (sns.heatmap (data.isnull(), yticklabels=False, cbar=False, cmap='viridis')): This line utilizes the seaborn library (sns) to create a heatmap. The heatmap visualizes missing values (NaN) in the DataFrame data.



From the visualization, we can observe that there are no missing (blank) values in our dataset. However there are misplaced or unnecessary values which we are going to handle.

PREPROCESS AND CLEAN DATA

Handling Missing Values (data.replace("?", pd.NA, inplace=True)): This addresses missing value representation. It replaces "?" in the DataFrame data with pd.NA (Not Available), a standard way to represent missing data in pandas. This makes handling missing values easier for subsequent steps.

Dropping Rows with Missing Values (data.dropna(inplace=True)): This method removes rows containing any missing values (NaN).

Encoding Categorical Features: Inside the loop, a LabelEncoder object is created. This encoder will be used to convert textual categories in the current column to numerical labels.

Visualizing Target Variable Distribution: This line creates a bar chart to visualize the distribution of the target variable (data['income']).

Separating Features and Target Variable [X = data.drop('income', axis=1)]: This line separates the features (independent variables) from the target variable (dependent variable). y=data['income']: This line extracts the target variable 'income' from the DataFrame data and stores it in a separate variable named y.

Normalizing Numerical Features: scaler = StandardScaler(): This line creates an instance of the StandardScaler class. This class is used to standardize the features by removing the mean and scaling them to unit variance.

Splitting the Dataset into Training and Testing Sets: This line uses the train_test_split() function from sklearn.model_selection to divide the data into training and testing sets. After this step, you have the necessary data preparations completed:

- X_train: The features for the training set.
- X_test: The features for the testing set.
- y_train: The target variable values for the training set.
- y_test: The target variable values for the testing set.

TRAIN THE MODEL

This section covers the training of the model using each of the four selected algorithms, namely, Logistic Regression, Decision Tree, Random Forest Classifier, and Support Vector Machine model.

1. Initialize the model: in this step we initialize an instance of the algorithm object.

- logistic_model = LogisticRegression(random_state=42, max_iter=1000)
- decision_tree_model = DecisionTreeClassifier(random_state=42)
- random_forest_model = RandomForestClassifier(random_state=42, n_estimators=100)
- svm_model = SVC(kernel='linear', random_state=42)

2. Train the model: in this step we train the model on the training dataset.

- logistic_model.fit(X_train, y_train)
- decision_tree_model.fit(X_train, y_train)
- random_forest_model.fit(X_train, y_train)
- svm_model.fit(X_train, y_train)

3. Make Prediction: in this step, we make prediction on the test set.

- y_pred = logistic_model.predict(X_test)
- y_pred = decision_tree_model.predict(X_test)
- y_pred = random_forest_model.predict(X_test)
- y_pred = svm_model.predict(X_test)

EVALUATION

1. Confusion Matrix: A confusion matrix is a table that summarizes the performance of a classification model. It shows the number of correct and incorrect predictions made by the model for each class. Here are the terms used in a confusion matrix: True Positive (Correctly predicted positive cases), False Positive (Incorrectly predicted positive cases (Type I error)), True Negative (Correctly predicted negative cases), False Negative (Incorrectly predicted negative cases (Type II error)). By analyzing the confusion matrix, you can gain insights into the strengths and weaknesses of your model.

2. Accuracy score: is the ratio of correctly predicted instances to the total number of instances in the dataset. Accuracy is a metric that provides an overall measure of a model's performance.

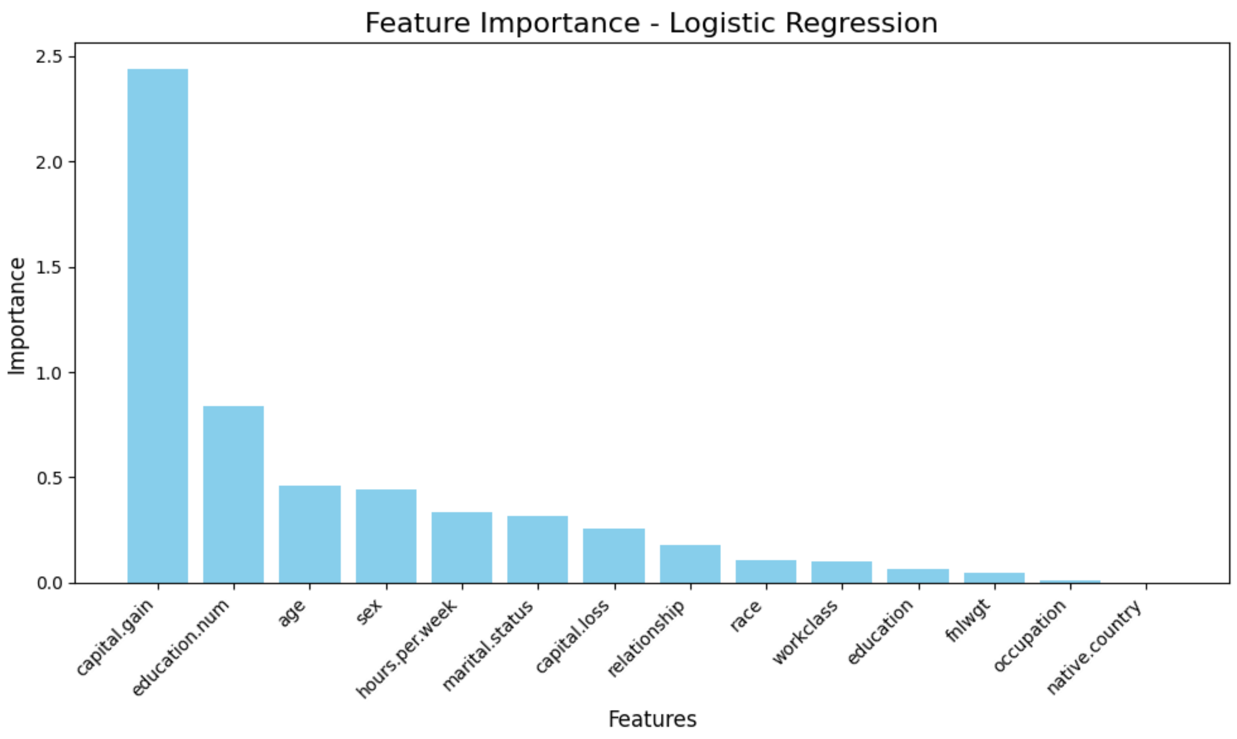
- **accuracy = accuracy_score(y_test, y_pred)** to calculate the accuracy score
- **print(f"Accuracy: {accuracy:.4f}\n")** to print the calculated accuracy score

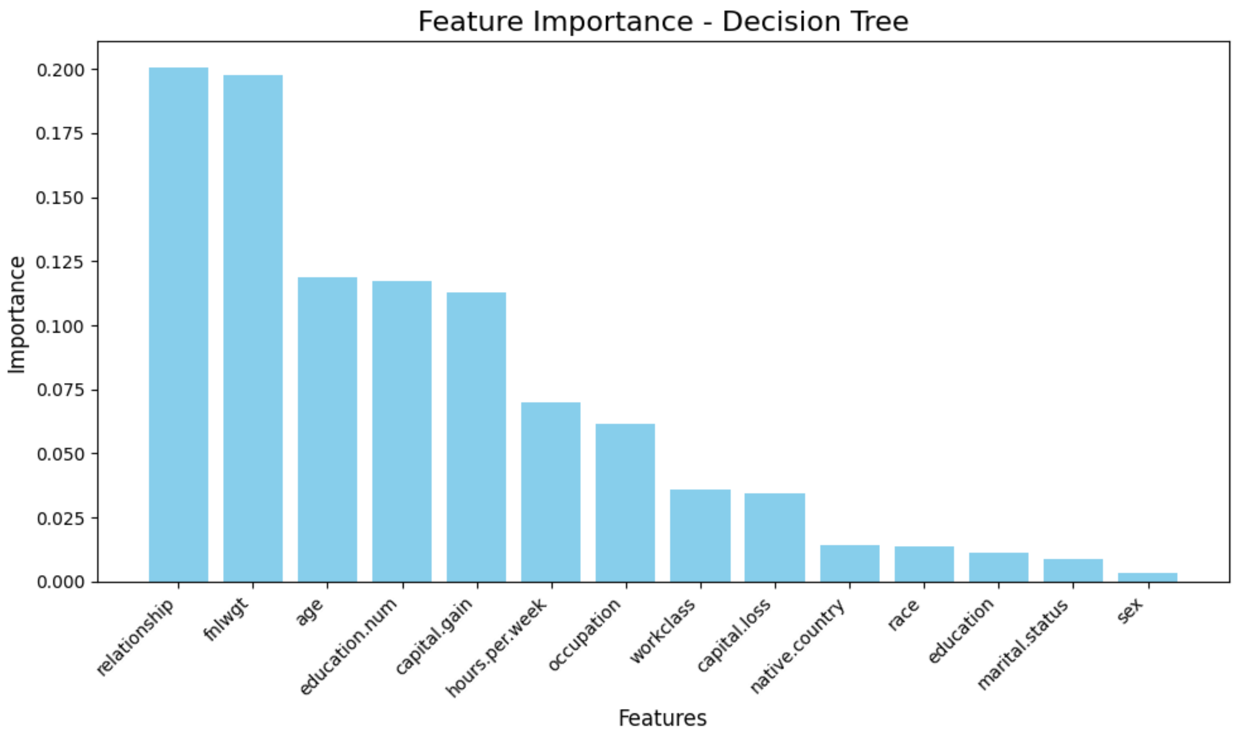
3. Classification Report: this gives us the value of precision, recall and f-1 score.

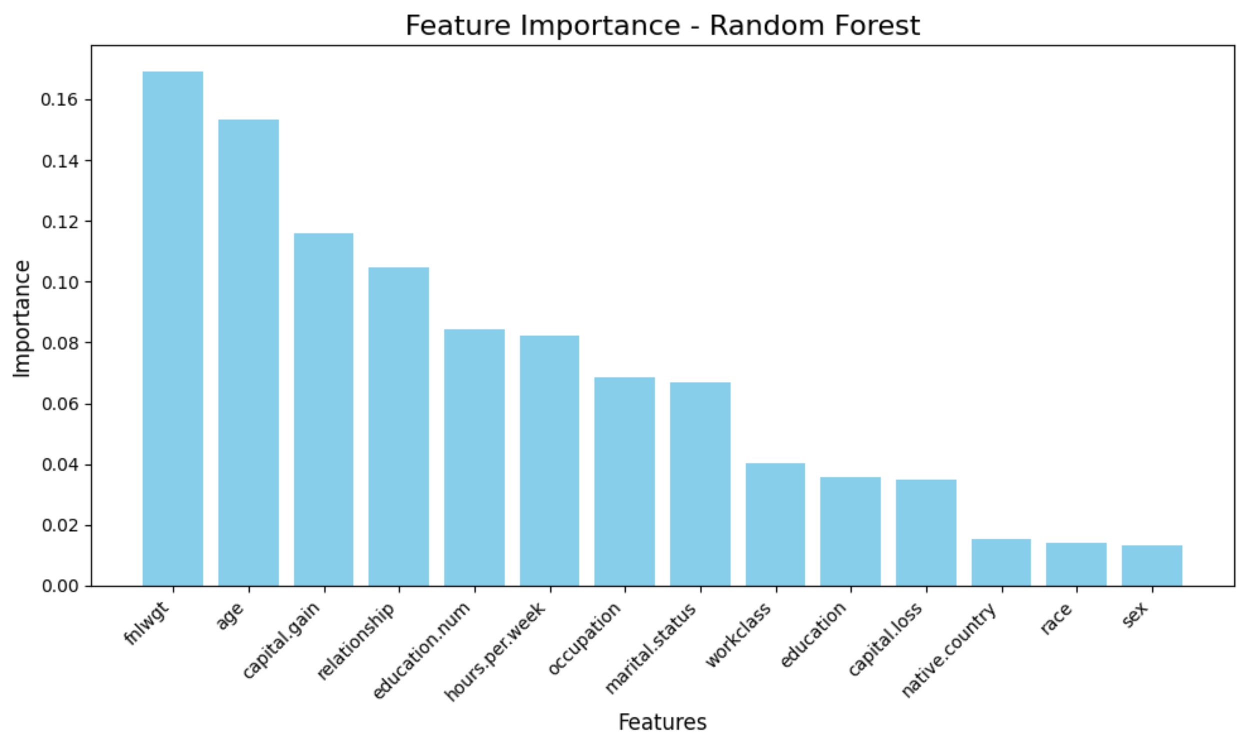
- class_report = classification_report(y_test, y_pred)
- print(class_report)

FEATURE IMPORTANCE

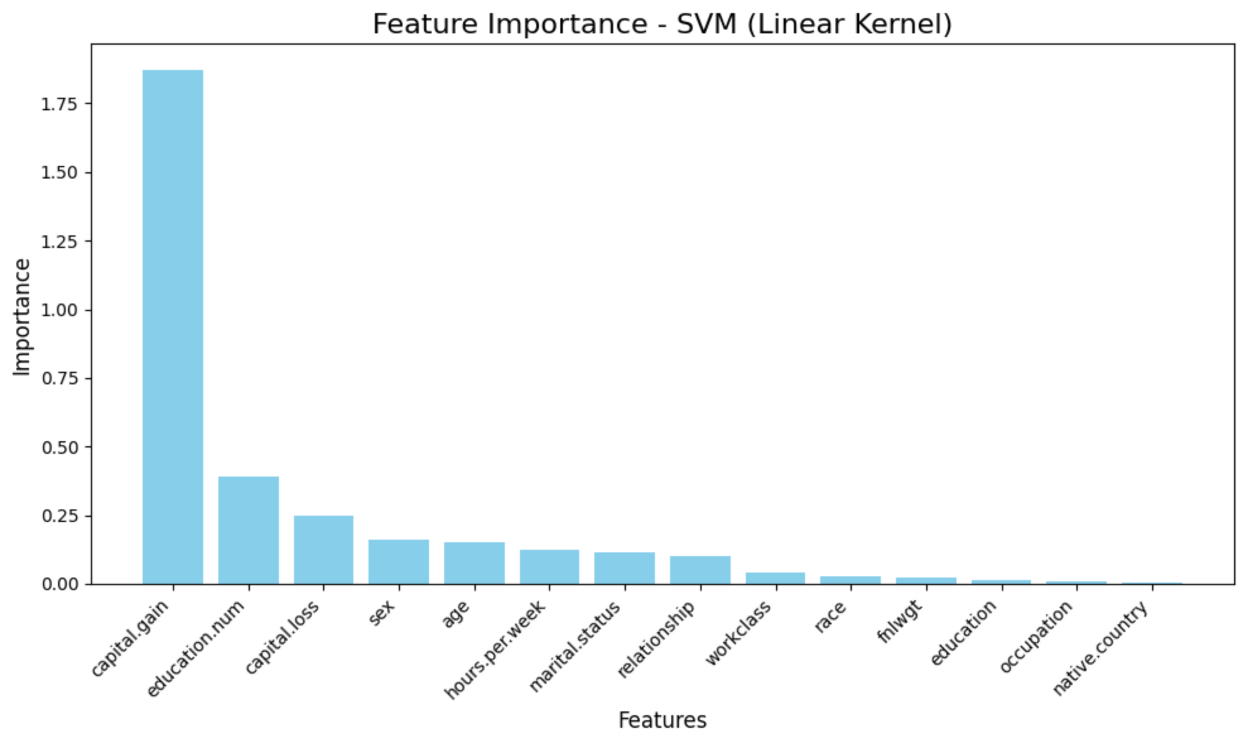
This section helps to understand which features (independent variables) of the dataset have the most contribution to the target variable for each algorithm







Feature Importance - Random Forest
Importance
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
fnlwgt
age
capital.gain
relationship
education.num
hours.per.week
occupation
marital.status
workclass
education
capital.loss
native.country
race
sex
Features



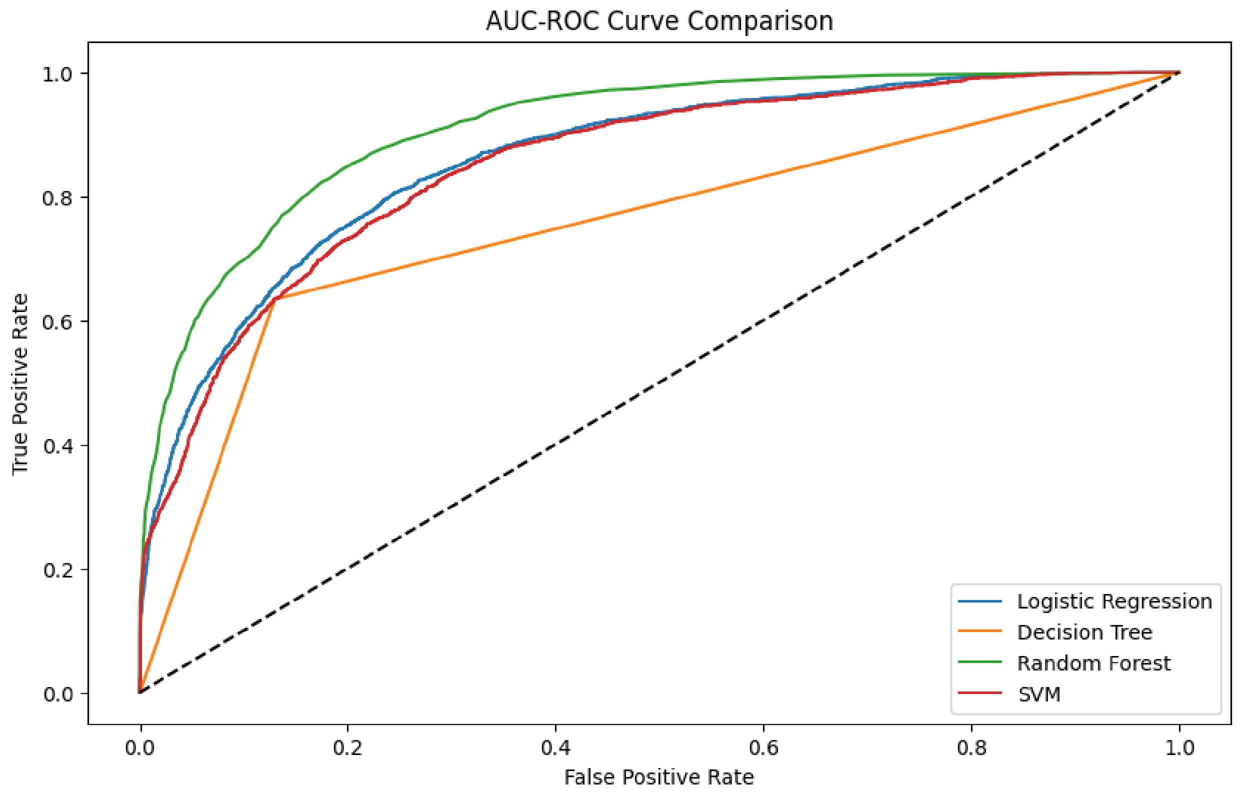
Feature Importance - SVM (Linear Kernel)
Importance
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
capital.gain
education.num
capital.loss
sex
age
hours.per.week
marital.status
relationship
workclass
race
fnlwgt
education
occupation
native.country
Features

EVALUATION OF THE MODELS (COMPARISON)

Using the above stated performance metrics, comparing the performance of each algorithms can be conducted. The following **table** shows the value of the performance metrics for each model, namely, Logistic Regression, Decision Tree, Random Forest and SVM. The performance metrics used are Accuracy, Precision, Recall, F-1 score and AUC-ROC.

	Accuracy	**Precision**	**Recall**	**F-1 score**	**AUC-ROC**
Logistic Regression	0.829926	0.752658	0.471581	0.579853	0.861148
Decision Tree	0.811471	0.618284	0.633659	0.625877	0.752021
Random Forest	0.859653	0.761448	0.634991	0.692494	0.909121
SVM	0.809703	0.805300	0.310391	0.448077	0.851633

Moreover, here is the **visual representation** of the **AUC-ROC curve comparison**:



Model Comparison by Performance Metrics

Accuracy:

- Best Model: **Random Forest** (Accuracy: 0.859653)
- Random Forest has the highest accuracy, indicating it performs best at correctly classifying all instances overall.

Precision:

- Best Model: **SVM** (Precision: 0.805300)
- SVM is the most precise, meaning it minimizes false positives better than other models.

Recall:

- Best Model: **Random Forest** (Recall: 0.634991)
- It means that Random Forest excels at capturing true positives, which is critical when identifying all positive cases is important.

F1-Score:

- Best Model: **Random Forest** (F1-Score: 0.692494)
- It means that Random Forest strikes the best balance between precision and recall, as indicated by the highest F1-score.

AUC-ROC:

- Best Model: **Random Forest** (AUC-ROC: 0.909121)
- It means that Random Forest provides the best separation between the positive and negative classes across all thresholds.

Overall Model Performance Ranking

Random Forest is the best-performing model across the majority of metrics and is recommended for this problem. Logistic Regression and SVM also show strengths in specific metrics but are not as good as Random Forest. Decision Tree ranks the lowest in terms of overall performance. The models can be ranked as follows:

1. Random Forest: Excels in most metrics (Accuracy, Recall, F1-Score, AUC-ROC) and provides a well-rounded performance.

2. Logistic Regression: Performs well in terms of AUC-ROC and has relatively good accuracy.

3. SVM: Offers the best precision but low with recall and F1-score, indicating an imbalance in handling false negatives.

4. Decision Tree: Although reasonable in recall and F1-score, it lags in accuracy and AUC-ROC compared to other models.

CONCLUSION

In this project, various machine learning classification algorithms were implemented to predict whether an individual earns more than $50,000 annually based on demographic and occupational features. Four algorithms, namely, Logistic Regression, Decision Tree, Random Forest, and Support Vector Machine (SVM), were evaluated using key performance metrics: accuracy, precision, recall, F1-score, and AUC-ROC.

Among the models, Random Forest emerged as the best-performing algorithm, consistently achieving high scores across all metrics, including accuracy, recall, F1-score, and AUC-ROC. Logistic Regression and SVM demonstrated strengths in specific metrics such as AUC-ROC and precision, respectively, while Decision Tree provided reasonable performance but fell short compared to the other models.

This analysis highlights the value of methods like Random Forest for classification tasks involving diverse and complex datasets. The findings underline the importance of considering multiple performance metrics to select the most suitable model for a given problem.